

AD-A142 355

A SECURITY MODEL FOR MILITARY MESSAGE SYSTEMS(U) NAVAL  
RESEARCH LAB WASHINGTON DC C E LANDWEHR ET AL.  
31 MAY 84 NRL-8806

1/1

UNCLASSIFIED

F/G 9/2

NL

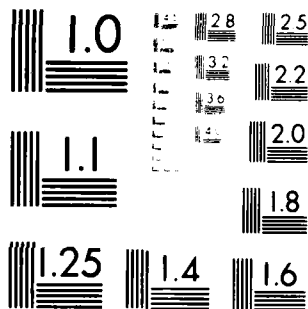
END

DATE

FILED

8 84

DTIC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

2

NRL Report 8806

# A Security Model For Military Message Systems

CARL E. LANDWEHR, CONSTANCE L. HEITMEYER, AND JOHN MCLEAN

*Computer Science and Systems Branch  
Information Technology Division*

AD-A142 355

May 31, 1984

DTIC FILE COPY



DTIC  
ELECTE  
JUN 21 1984  
S B

NAVAL RESEARCH LABORATORY  
Washington, D.C.

Approved for public release; distribution unlimited.

84 06 21 180

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		1b. REPORT FILE MARKING		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE				
4. PERFORMING ORGANIZATION REPORT NUMBER <b>NRL Report 8806</b>		5. MONITORING ORGANIZATION REPORT NUMBER		
6a. NAME OF PERFORMING ORGANIZATION <b>Naval Research Laboratory</b>	6b. OFFICE SYMBOL <i>If applicable</i> <b>Code 7593</b>	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State and ZIP Code) <b>Washington, DC 20375</b>		7b. ADDRESS (City, State and ZIP Code)		
8a. NAME OF FUNDING SPONSORING ORGANIZATION <b>Office of Naval Research</b>	8b. OFFICE SYMBOL <i>If applicable</i>	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State and ZIP Code) <b>Arlington, VA 22217</b>		10. SOURCE OF FUNDING NOS.		
11. TITLE (Include Security Classification) <b>(See Page ii)</b>		PROGRAM ELEMENT NO. <b>61153N</b>	PROJECT NO. <b>RR0140941</b>	TASK NO. <b>75-0113-0-3</b>
12. PERSONAL AUTHOR(S) <b>Landwehr, Carl E.; Heitmeyer, Constance L.; and McLean, John</b>				
13a. TYPE OF REPORT <b>Final</b>	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Yr. Mo. Day) <b>1984 May 31</b>		15. PAGE COUNT <b>24</b>
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB GR		
09	02		Computer security	Message systems
15	03		Security requirements	Multilevel security
			Security model	Confinement
			Formal models	Software engineering
			(See Page ii)	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Military systems that process classified information must operate in a secure manner, i.e., they must protect information adequately against unauthorized disclosure, modification, and withholding. A goal of current research in computer security is to facilitate the construction of multilevel secure systems, systems that protect information of various classifications from users with different clearances. Security models are used to define the concept of security embodied by a computer system. A single model, called the Bell-LaPadula model, has dominated recent efforts to build secure systems but has deficiencies. We are developing a new approach to defining security models based on the idea that a security model should be derived from a specific application. To evaluate our approach, we have formulated a security model for a family of military message systems. This report introduces the message-system application, describes the problems of using the Bell-LaPadula model in real applications, presents our security model both informally and formally, and summarizes our approach to developing secure message systems. Significant aspects of the security model are its definition of multilevel objects and its inclusion of application-dependent security assertions. Prototypes based on this model are being developed.				
20. DISTRIBUTION AVAILABILITY OF ABSTRACT UNCLASSIFIED UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input checked="" type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		
22a. NAME OF RESPONSIBLE INDIVIDUAL <b>Carl E. Landwehr</b>		22b. TELEPHONE NUMBER <i>Include Area Code</i> <b>(202) 767-3381</b>	22c. OFFICE SYMBOL <b>7593</b>	

11. TITLE (Include Security Classification)

A Security Model for Military Message Systems

18. SUBJECT TERMS (Continued)

Formal specification  
Verification

**DTIC**  
**ELECTE**  
**S** JUN 21 1984 **D**  
**B**



Accession For	
DTIC GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

## CONTENTS

INTRODUCTION .....	1
REQUIREMENTS OF MILITARY MESSAGE SYSTEMS .....	2
Functional Requirements .....	2
Security Requirements .....	2
EXPERIENCE WITH THE BELL-LAPADULA MODEL AND TRUSTED PROCESSES .....	3
The Military Message Experiment (MME) .....	3
AFDSC Multics .....	4
Kernelized Secure Operating System (KSOS) .....	5
Guard .....	5
Lessons Learned .....	5
MILITARY MESSAGE SYSTEM (MMS) SECURITY MODEL .....	6
Definitions .....	6
User's View of MMS Operation .....	8
Security Assumptions .....	8
Security Assertions .....	9
Discussion .....	10
FORMALIZING THE MMS SECURITY MODEL .....	13
Approach .....	14
System State .....	14
Secure System .....	16
Discussion .....	18
Correspondence to the Informal Model .....	19
Storage Channels .....	19
A Basic Security Theorem for the Formal MMS Security Model .....	20
OTHER ELEMENTS OF OUR APPROACH .....	20
Family Methodology .....	20
Requirements Document .....	21
Rapid Prototyping .....	21
CONCLUSIONS .....	22
ACKNOWLEDGMENTS .....	22
REFERENCES .....	22

# A SECURITY MODEL FOR MILITARY MESSAGE SYSTEMS

## INTRODUCTION

A system is *secure* if it protects information that it processes adequately against unauthorized disclosure, unauthorized modification, and unauthorized withholding (also called denial of service). We say "adequately," because no practical system can achieve these goals without qualification; security is inherently relative. A secure system is *multilevel secure* if it protects information of different classifications from users with different clearances; thus, some users may not be cleared for all of the information that the system processes.

In recent years, security models have been developed both to describe the protection that a computer actually provides and to define the security rules it is required to enforce [1]. In our view, a security model should enable users to understand how to operate the system effectively, implementers to understand what security controls to build, and certifiers to determine whether the system's security controls are consistent with relevant policies and directives and whether these controls are implemented correctly [2,3].

Recently the Bell-LaPadula model [4,5] has dominated efforts to build secure systems. The publication of this model advanced the technology of computer security by providing a mathematical basis for examining the security provided by a given system. Moreover, the model was a major component of one of the first disciplined approaches to building secure systems.

Unfortunately, a system that strictly enforces the axioms of the Bell-LaPadula model without the use of *trusted subjects* is often impractical: in real systems, users may need to perform operations that, although they do not violate our intuitive concept of security, do violate one of the assertions (the \*-property) of the model. For example, a user may need to extract an unclassified paragraph from a confidential document and use it in an unclassified document. A system that strictly enforces the Bell-LaPadula model would prohibit this operation unless it were performed by a trusted subject. Consequently, systems based on this model usually contain mechanisms that permit some operations that the axioms prohibit, e.g., the *trusted processes* in KSOS [6] and SIGMA [7]. The presence of such mechanisms makes it difficult to determine the actual security policy enforced by the system and complicates the user interface.

To avoid these problems, we take a different approach. We believe that a security model should be derived from a specific application. To evaluate our approach, we have formulated a security model for a family of military message systems. Defining an application-based security model is part of a larger effort whose goals are to develop a disciplined approach to the production of secure systems and to produce fully worked out examples of a requirements document and a software design for such systems. In this report, we introduce the message-system application, discuss the Bell-LaPadula trusted process approach to building secure systems, present a security model for military message systems both informally and formally, compare our model with the Bell-LaPadula model, and summarize our approach to building secure message systems.

## REQUIREMENTS OF MILITARY MESSAGE SYSTEMS

In recent years, automation has been applied increasingly to the handling of military messages [8]. While the primary purpose of military message systems is to process *formal messages*, i.e., official messages exchanged by military organizations, such systems may also handle *informal messages*, i.e., unofficial messages exchanged by individuals. Formal messages are transmitted over military networks, such as AUTODIN; their format and use is governed by military standards. Examples of informal messages are those currently supported by several message systems (e.g., HERMES [9]) available on the ARPA network.

### Functional Requirements

Message-system operations may be organized into three categories: operations on incoming messages, operations on outgoing messages, and message storage and retrieval. Operations in the first category permit a user to display and print messages he has received. Second-category operations support the creation, editing, and transmission of outgoing messages. Message storage and retrieval operations allow users to organize messages into message files and to retrieve messages via single keys (e.g., message id) or combinations of keys (e.g., subject and originator). Typically, military systems that process formal messages provide the same operations as systems that process informal messages plus several additional operations, such as distribution determination, action and information assignment, and release [8].

### Security Requirements

Each formal military message is composed of several fields, including *To*, *From*, *Info*, *Date-Time-Group*, *Subject*, *Text*, *Security*, and *Precedence*. A classification, such as *Unclassified* or *Secret*, is assigned to each field and to some subfields, e.g., the paragraphs of the *Text* field; further, the overall message has a classification that is at least as high as that of any field or subfield. Thus, the *Subject* field of a message may be classified at a lower level than the message as a whole, and two paragraphs of the *Text* field may have different classifications.

In some data-processing applications, users process information at a single security level for long periods of time. In contrast, message-system users often need to handle data of several classifications during a single computer session. For example, a user may wish to compose an unclassified message based in part on a previous classified message he has received. To accomplish this, he must simultaneously display the classified information and compose the unclassified message. As a further example, the user may wish to scan newly arrived messages and print only those that are unclassified. To do so, he must display data of several different classifications and then print a hard copy of only the unclassified data.

Military message systems are required to enforce certain security rules. For example, they must ensure that users cannot view messages for which they are not cleared. Unfortunately, most automated systems cannot be trusted to enforce such rules. The result is that many military message systems operate in "system-high" mode: each user is cleared to the level of the most highly classified information on the system. A consequence of system-high operation is that all data leaving the computer system must be classified at the system-high level until a human reviewer assigns the proper classification.

A goal of our research is to design message systems that are multilevel secure. Unlike systems that operate in system-high mode, multilevel secure systems do not require all users to be cleared to the level of the highest information processed. Moreover, information leaving such a system can be assigned its actual security level rather than the level of the most highly classified information in the system. Unlike a system that operates at system-high, a multilevel system can preserve the different classifications of information that it processes.



## EXPERIENCE WITH THE BELL-LAPADULA MODEL AND TRUSTED PROCESSES

Although its complete formal statement is lengthy and complex, the Bell-LaPadula model may be briefly summarized by the following two axioms:

- the *simple security rule*, which states that a subject cannot read information for which it is not cleared ("no read up"), and
- the *\*-property*, which states that a subject cannot move information from an object with a higher security classification to an object with a lower classification ("no write down").

These axioms are to be enforced by restricting the access rights that subjects (e.g., users and processes) have to objects (e.g., files and devices).

A less frequently described part of the Bell-LaPadula model is its concept of *trusted subjects*, i.e., subjects that are allowed "to operate without the extra encumbrance of the \*-property," because they are trusted "never [to] mix information of different security levels" [10]. More precisely, a trusted subject can have simultaneous read access to objects of classification  $x$  and write access to objects of classification  $y$ , even if the classification of  $y$  is lower than the classification of  $x$ . The formal statement of the Bell-LaPadula model places no constraints on the trusted subject's violations of the \*-property.

A number of projects have used the Bell-LaPadula model to describe their security requirements. In these projects, strict enforcement of the axioms of Bell-LaPadula without trusted subjects has proved to be overly restrictive, since users frequently need to perform operations that do not violate security but do violate the \*-property. To deal with this problem, *trusted processes* were introduced as an implementation of the concept of trusted subjects. In the following sections, we summarize experience with the Bell-LaPadula model and trusted processes in four projects: the Military Message Experiment (MME), the Air Force Data Services Center (AFDSC) Multics, the Kernelized Secure Operating System (KSOS), and the Guard message filter.

### The Military Message Experiment (MME)

The MME's goal was to evaluate the utility of an interactive message system in an operational military environment [11]. During the MME, more than 100 military officers and staff personnel used SIGMA, the message system developed for the experiment, to process their messages [12,13]. Although SIGMA was built on the nonsecure TENEX operating system, its user interface was designed as though it were running on a *security kernel*, i.e., a minimal, tamperproof mechanism that assures that all accesses that subjects have to objects conform to a specified security model. SIGMA's user interface was designed so that it would not change if SIGMA were rebuilt to operate with a security kernel.

During the planning phase of the MME, it was decided that SIGMA would enforce the Bell-LaPadula model [7]. This decision led to a number of difficulties, three of which are described here. The first problem arose from the initial decision, later changed, to adopt the model without trusted subjects; the other two problems apply to Bell-LaPadula with or without trusted subjects.

#### *Prohibition of Write-Downs*

The \*-property of Bell-LaPadula disallows write-downs; yet, in certain cases, message-system users need to lower the classification of information. For example, a user may create a message at top secret and, after he has entered the message text, decide that the message classification should be secret. A system that strictly enforces the \*-property would prohibit a user from reducing the message classification. The user would be required to create a new message at secret and reenter the text.

### *Absence of Multilevel Objects*

Bell-LaPadula recognizes only single-level objects; some message system data objects (e.g., messages and message files) are inherently multilevel. A computer system that treats a multilevel object as single-level can cause some information to be treated as more highly classified than it really is. For example, when a user of such a system extracts an unclassified paragraph from a secret message, the system labels the paragraph secret even though the paragraph is actually unclassified.

### *No Structure for Application-Dependent Security Rules*

Military message systems must enforce some security rules that are absent in other applications. An example is a rule that allows only users with release authority to invoke the release operation. (This release of a message is security-relevant because it allows a wider set of users to view the message and because it certifies that a particular military organization originated the message.) Such application-dependent rules are not covered by Bell-LaPadula and, hence, must be defined outside of it.

To address the first problem (and, to some extent, the third), the SIGMA developers designed a trusted process that is not constrained by the \*-property and is, therefore, permitted to perform write-downs. For example, a SIGMA user could search a file containing both unclassified and secret messages and then display an unclassified message whose citation was returned by the search; such an operation required the intervention of the trusted process, since the message citation was transmitted from the secret process that did the search to the unclassified process that handled the message display. Unlike the Bell-LaPadula model, which puts no explicit constraints on write-downs performed by the trusted subjects, SIGMA's trusted process narrowly limited the cases in which write-downs were permitted. Reference 7 gives further details on the role of the trusted process in SIGMA.

SIGMA's use of a trusted process was helpful in that it relaxed the rigid constraints of Bell-LaPadula, thus permitting users to perform required operations. However, adding the trusted process also caused a serious problem: it made the security policy that SIGMA enforced difficult to understand. Interviews held during the MME revealed that few SIGMA users clearly understood the security policy that was being enforced. It was an assumption of SIGMA's design that user confirmation of security-relevant operations would prevent security violations; because users issued confirmations without comprehending why these confirmations were needed, this assumption was unwarranted.

### **AFDSC Multics**

In the mid-1970s, Multics was modified to include the *access isolation mechanism* (AIM). This version of Multics, which has been used at the AFDSC for several years, supports the assignment of security levels to processes and segments and enforces the Bell-LaPadula model. Multics-AIM also contains trusted functions, invoked via a special operating system gate, to enforce access control on objects smaller than a segment, to allow security officers to downgrade files in response to user requests, and to provide other "privileged" operations.

Although Multics-AIM is generally considered a success, experience with it at the AFDSC illustrates some difficulties that arise from strict enforcement of the Bell-LaPadula axioms and from the use of trusted functions. For example, if a user operating at the top secret level wishes to send an unclassified message to another user operating at the secret level, Multics-AIM requires that the message be treated as though it were top secret. The recipient is not notified of its arrival until he logs in as a top secret user.

Problems also occur when a user operating at a low security level tries to send mail to a user at a higher level. Mailbox segments in Multics-AIM are special: they have both a minimum and maximum

access level. The minimum is defined by the level of the directory that contains the mailbox segment. Thus, a user operating at unclassified is prohibited from sending a message to a mailbox located in a secret directory. In this case, the mail could not be sent unless the sender were to log out and log back in at the secret level. Because this situation arises frequently, system administrators are allowed to invoke a trusted function that permits them to send mail without logging out and logging back in again.

### **Kernelized Secure Operating System (KSOS)**

KSOS [6] was to be a security-kernel based system with a UNIX-compatible program interface on a DEC PDP-11. The KSOS security kernel was designed to enforce strictly the axioms of the Bell-LaPadula model on user-provided programs. To handle those situations where strict enforcement is incompatible with functional requirements, the kernel recognizes certain privileges that allow some processes to circumvent parts of this enforcement. These privileges include the ability to violate the \*-property, to change the security or integrity level of objects, and to invoke certain security kernel functions.

KSOS developers defined a special category of software, called *non-kernel security related* (NKSr), that supports such privileges. For example, the "Secure Server" of the KSOS NKSr allows a user to change the security level of files he owns and to print a file classified at a lower security level without raising the security level of the printed output to the level of his process. Both of these operations would be prohibited by strict enforcement of the Bell-LaPadula axioms.

### **Guard**

The Guard message filter [14] is a computer system that supports the monitoring and sanitization of queries and responses between two database systems operating at different security levels. When a user of the less-sensitive system requests data from the more-sensitive system, a human operator of the Guard must review the response to ensure that it contains only data that the user is authorized to see. The operator performs this review via a visual display terminal.

One version of the Guard is being built on a security kernel that enforces the axioms of the Bell-LaPadula model. However, strict enforcement of the \*-property is not possible, since a major requirement of the Guard system is to allow the operator to violate it, i.e., to allow information from the more sensitive system to be sanitized and "downgraded" (or simply downgraded), so that it can be passed to systems that store less-sensitive information. An important component of this version's design is the trusted process that performs this downgrading.

### **Lessons Learned**

Experience has shown that, on the one hand, the axioms of the Bell-LaPadula model are overly restrictive: they disallow operations that users require in practical applications. On the other hand, trusted subjects, the mechanism provided to overcome some of these restrictions, are not restricted enough. The formal model provides no constraints on how trusted subjects violate the \*-property. Consequently, developers have had to develop *ad hoc* specifications for the desired behavior of trusted processes in each individual system. While such an approach relaxes the rigid enforcement of the \*-property, it introduces two additional problems:

- Use of the axioms in conjunction with trusted processes makes it difficult to determine the exact nature of the security rules that a system enforces. In the MME and the other three projects described, the security rules enforced by the system as a whole are not the same as the axioms of the model. The actual security rules enforced by each system include both the axioms of the Bell-LaPadula model and the exceptions allowed by the trusted processes.

- Because the actual policies in practical systems deviate from the Bell-LaPadula axioms, any inductive proof that such a system maintains a secure state, based on strict enforcement of the axioms of the model, is a proof about only part of the system and cannot apply to the entire system.

Moreover, trusted subjects do not address the two other problem areas of the Bell-LaPadula model discussed above, i.e., its failure to support multilevel objects and its lack of a structure for including application-dependent security rules.

## MILITARY MESSAGE SYSTEM (MMS) SECURITY MODEL

Our goal is to define a single, integrated security model that captures the security policy that a military message system must enforce, without mentioning the techniques or mechanisms used to implement the system or to enforce the policy. The security model defined here is intended to allow users to understand security in the context of message systems, to guide the design of military message systems, and to allow certifiers to evaluate such systems. The model presented here is informal; it is the basis for the formal model presented in the following section.

In this section we define some terms, use them to describe how a user views the system's operation, and state assumptions and assertions, based on the terms and the user's view of operation, that are intended to be sufficient to assure the security of the system. The security model comprises the definitions, the user's view of operation, the assumptions, and the assertions. It is a revision of earlier work [3].

This model does not address auditing, although secure message systems clearly require auditing mechanisms. The existence of an audit trail may deter potential penetrators, but auditing is primarily a technique for providing accountability and for detecting security violations after the fact. The security model focuses on assertions that, if correctly enforced, will prevent security violations. Consequently, assertions and assumptions about auditing do not appear; in a more detailed system specification, auditing requirements would be explicit.

The model itself places no constraints on the techniques used to implement a military message system or to verify that a particular system correctly enforces the assertions of the model. An implementation based on a complete formal specification and proof of correctness would be as admissible as one based on a security kernel and trusted processes, or even one employing standard software engineering techniques for design, testing, and validation. By separating the statement of the security model from the concerns of implementation and verification, we can allow for advances in these areas without depending on them.

### Definitions

The definitions given here correspond in most cases to those in general use and are given here simply to establish an explicit basis for the model. We distinguish between "objects," which are single-level, and "containers," which are multilevel. We also introduce the concept of "user roles," which define job-related sets of privileges.

**Classification:**\* A designation attached to information that reflects the damage that could be caused by unauthorized disclosure of that information. A classification includes a sensitivity level (unclassified, confidential, secret, or top secret) and a set of zero or more compartments (crypto, nuclear, etc.). The set of classifications, together with the relation

\*This definition corresponds to that used by other authors for security level. In this report, security level and classification are synonyms.

defining the allowed information flows between levels, forms a lattice [15]. Most dissemination controls, such as NATO, NOFORN, and NOCONTRACTOR, can be handled as additional compartment names.

- Clearance:** The degree of trust associated with a person. This is established on the basis of background investigations and the tasks performed by the person. It is expressed in the same way as classifications are, as a sensitivity level and a (possibly null) compartment set. In a secure MMS, each user will have a clearance, and operations performed by the MMS for that user will check the user's clearance and the classifications of objects to be operated on to determine whether the required level of access is authorized. Some other characteristics of a user, such as his nationality and employer, may also be treated as part of his clearance so that dissemination controls are handled properly.
- UserID:** A character string used to denote a user of the system. To use the MMS, a person must present a userID to the system, and the system must authenticate that the user is the person corresponding to that userID. This process is called logging in. Since clearances are recorded on the basis of one per userID, each user should have a unique userID.
- User:** A person who is authorized to use the MMS.
- Role:** The job the user is performing, such as downgrader, releaser, or distributor. A user is always associated with at least one role at any instant, and the user can change roles during a session. To act in a given role, the user must be authorized for it. Some roles may be assumed by only one user at a time (e.g., distributor). With each role comes the ability to perform certain operations.
- Object:** A single-level unit of information. An object is the smallest unit of information in the system that has a classification. An object thus contains no other objects — it is not multilevel. There are many kinds of objects; an example is the date-time group of a message.
- Container:** A multilevel information structure. A container has a classification and may contain objects (each with its own classification) and/or other containers. In most MMS family members, message files and messages are containers. Some fields of a message (such as the *Text* field) may be containers as well. The distinction between an object and a container is based on type, not current contents: within a family member, if an entity of type message file is a container, then all message files in that family member are containers, even if some of them are empty or contain only objects and/or containers classified at the same level as the message file itself. Devices such as disks, printers, tape drives, network interfaces, and users' terminals will be containers, rather than objects, in most MMS family members.
- Entity:** Either a container or an object.
- Container clearance required (CCR):** An attribute of some containers. For some containers, it is important to require a minimum clearance, so that if a user does not have at least this clearance, that user cannot view any of the entities within the container. Such containers are marked with the attribute *container clearance required* (CCR). For example, a user with only a confidential clearance could be prohibited from viewing just the confidential paragraphs of a message classified top secret if the message (which is a container) is marked CCR. On the other hand, given a message file containing both

top secret and confidential messages, it may be acceptable to allow the user in question to view the confidential ones even though the container (message file) as a whole is classified top secret. In this case, the file would *not* be marked CCR

**ID:** Identifier. An ID names an entity without referring to other entities. For example, the name of a message file is an ID for that file. Some, but not necessarily all, entities can be named by identifiers. Entities may also be named in other ways, e.g., "the current message's *Text* field's third paragraph." (This is an indirect reference, defined below.)

**Direct reference:** A reference to an entity is direct if it is the entity's ID

**Indirect reference:** A reference to an entity is indirect if it is a sequence of two or more entity names (of which only the first may be an ID).

**Operation:** A function that can be applied to an entity. It may simply allow that entity to be viewed (e.g., display a message), or it may modify the entity (update a message), or both (create a message). Some operations may involve more than one entity (copy a message from one message file to another).

**Access Set:** A set of triples (userID or role, operation, operand index) that is associated with an entity. The operations that may be specified for a particular entity depend on the type of that entity. If a given operation requires more than one operand, the operand index specifies the position in which a reference to this entity may appear as an operand. For messages, operations include DISPLAY, UPDATE, DELETE, and others. The existence of a particular triple in the access set implies that the user corresponding to the specified userID or role is authorized to invoke the specified operation on the entity with which the set is associated.

**Message:** A particular type implemented by an MMS. In most MMS family members, a message will be a container, though messages may be objects in some receive-only systems. A message will include *To*, *From*, *Date-Time-Group*, *Subject*, *Releaser*, and *Text* fields, and additional fields as well. A draft message also includes a *Drafter* field.

### User's View of MMS Operation

We present the following as a model of the use of a secure MMS. Terms defined in the previous section are printed in capitals.

Persons can gain access to the system only by logging in. To log in, a person presents a **USERID**; the system then performs authentication using passwords, fingerprint recognition, or any appropriate technique. Following a successful authentication, the **USER** invokes **OPERATIONS** to perform the functions of the message system. The **OPERATIONS** a **USER** may invoke depend on his **USERID** and the **ROLES** for which he is authorized; by applying **OPERATIONS**, the **USER** may view or modify **OBJECTS** or **CONTAINERS**. The system enforces the security assertions listed in the following sections (that is, it prevents the user from performing **OPERATIONS** that would contradict these assertions).

### Security Assumptions

It will always be possible for a valid user to compromise information to which he has legitimate access. To make the dependence of system security on user behavior explicit, we list the following

assumptions; these assumptions are really security assertions that can only be enforced by the users of the system

- Assumption 1. The System Security Officer (SSO) assigns clearances, device classifications, and role sets properly
- Assumption 2. The user enters the correct classification when composing, editing, or reclassifying information.
- Assumption 3. Within a classification, the user addresses messages and defines access sets for entities he creates so that only users with a valid need-to-know can view the information.
- Assumption 4. The user properly controls information extracted from containers marked CCR (i.e., exercises discretion in moving that information to entities that may not be marked CCR).

The basis for these assumptions is that, when there is no other source of information about the classification of an entity or the clearance of a person, the user is assumed to provide information that is correct.

### Security Assertions

The following statements hold for a multilevel secure MMS:

- |              |                                 |   |
|--------------|---------------------------------|---|
| Assertion 1. | Authorization                   | A user can only invoke an operation on an entity if the user's userID or current role appears in the entity's access set along with that operation and with an index value corresponding to the operand position in which the entity is referred to in the requested operation. |
| Assertion 2. | Classification hierarchy        | The classification of any container is always at least as high as the maximum classification of the entities it contains.   |
| Assertion 3. | Changes to objects              | Information removed from an object inherits the classification of that object. Information inserted into an object must not have a classification higher than the classification of that object.  |
| Assertion 4. | Viewing                         | A user can only view (on some output medium) an entity with a classification lower than or equal to the user's clearance and the classification of the output medium. (This assertion applies to entities referred to either directly or indirectly.)                           |
| Assertion 5. | Access to CCR entities          | A user can have access to an indirectly referenced entity within a container marked Container Clearance Required only if the user's clearance is greater than or equal to the classification of that container.   |
| Assertion 6. | Translating indirect references | A user can obtain the ID for an entity that he has referred to indirectly only if he is authorized to view that entity via that reference.  |

LANDWEHR, HEITMEYER, AND MCLEAN

Assertion 7.	Labeling requirement	Any entity viewed by a user must be labeled with its classification.
Assertion 8.	Setting clearances, role sets, device levels	Only a user with the role of System Security Officer can set the clearance and role set recorded for a userID or the classification assigned to a device. A user's current role set can be altered only by that user or by a user with the role of System Security Officer.
Assertion 9.	Downgrading	No classification marking can be downgraded except by a user with the role of downgrader who has invoked a downgrade operation.
Assertion 10.	Releasing	No draft message can be released except by a user with the role of releaser. The userID of the releaser must be recorded in the "releaser" field of the draft message.

### Discussion

Table 1 compares aspects of the MMS and Bell-LaPadula security models. The rest of this subsection clarifies the effects of the model in particular cases. The paragraphs here are not part of the model; the previous subsections define the model completely. Here we seek only to show how the model applies in specific circumstances.

- What prevents a user from copying a classified entity to an unclassified entity?

The classification of the entity being copied accompanies the data. Moving explicitly classified data to an unclassified container is a violation of Assertions 2 and 9 (unless the user requesting the operation is the downgrader and is performing a downgrade operation), since the classification of the data in question is effectively changed by the operation. Manipulations that affect only objects are covered by Assertion 3.

- What about copying a part of an object into another object?

A part of an object inherits the classification of the whole object (Assertion 3). Thus, moving part of an object into another object is disallowed by Assertions 2 and 3 unless the classification of the former object is lower than or equal to that of the latter. Note that this constraint does not affect the user's ability to remove an unclassified paragraph (an object) from a confidential document (a container) and use it in an unclassified document (another container).

- Does a user have a "log-in level" ?

*Log-in levels* are not explicitly part of the model, but the effect of a log-in level can be obtained through the classification of the user's terminal. The classification of the terminal is an upper bound on the classification of information that can be displayed on it (Assertion 4). If the user wishes to restrict further the level of the information that appears on the terminal, he may invoke an operation to reduce the classification of the terminal. The right to determine the classification of shared devices (disks, printers, etc.) will generally belong to the SSO. Note that restricting the level of the information that can appear on the user's terminal does not necessarily restrict the level of information that programs he invokes may have access to.

- Processes do not appear in the model but surely will be present in the implementation. How will their activities be constrained?



Table 1 — Comparison of Aspects of Bell-LaPadula and MMS Security Models

Parameter	Bell-LaPadula Model	MMS Model
Model components	Subjects, objects, accesses, access matrix, etc.	Users, entities, access sets, roles, etc.
Mechanism for changing state	Rules	System transform
Constraints on state changes	Axioms (simple security, *-property, etc.)	Security assertions (authorization, classification hierarchy, viewing, etc.)
Exceptions to constraints	Trusted subjects exempted from *-property enforcement	None — all users subject to same constraints
Structure for including application-dependent assertions	None — application considerations usually embodied in implementation of trusted subjects; separate sets of assertions may be specified to constrain behavior of implementation	Application-dependent security assertions included as part of basic security model
Structure for multilevel objects	None — included in practical systems through implementation of trusted subjects	Containers — integral part of security model
Structure for permitting write-downs, reclassification	*-property and other axioms prohibit these operations, but exceptions are permitted for trusted subjects	Restricted to users with role of system security officer via downgrade
Access control based on reference path	Only one reference path exists for each object	Direct and indirect references are distinguished, CCR mechanism allows separate control
Assumptions on security-relevant user behavior	Assumptions implicit	Assumptions explicit
Informal description of system behavior and security constraints	None	Included as informal model

Operations, rather than processes or programs, are in the model because they correspond more closely to the user's view of the system. To the user, the system offers functions that may be invoked by typing strings of characters, pushing function keys, etc. Each function can be understood by the user as an operation. In the implementation, processes are constrained to invoke only operations that preserve the truth of the assertions.

- Which entities in a particular message system will be containers and which will be objects?

This decision is part of the next more detailed level of the stated model. Some likely choices are that messages and message files will be containers and that the date-time group will be an object. It is not necessary that all message systems in the family make the same choices. If two message systems that make different choices communicate, some method of mapping between those entities that are objects in one system and containers in the other must be defined.

- How are entities created?

For each type of entity that users can create, there will be an operation that, when invoked, creates a new instance of that type. As with all other operations, only users who are authorized for it can invoke it. Thus, it is not necessarily the case that any particular user will be able to create any particular kind of object; he must be authorized to do so. In particular, only users authorized for certain roles may be allowed to create certain kinds of entities.

- How does a user refer to an object or a container?

Some entities have identifiers (IDs) that allow them to be named directly. A given entity may have zero, one, or more IDs. An entity may also be referred to indirectly by a qualified name (see the example under the definition of ID). A user (or a program he invokes) can refer to an entity using any valid ID or qualified name. The former is called a *direct reference* and the latter an *indirect reference*.

- What policy governs access to an entity in a container? Is the classification of the container or of the contents tested, and with what is it compared?

The answers to these questions depend on the type of access (the operation invoked) and whether the reference is direct or indirect. If the entity is referred to directly for viewing, Assertion 4 gives the appropriate restriction. If the reference is indirect, there are two cases, depending on whether or not the object is within a container marked CCR. If it is, both Assertions 4 and 5 have an effect; otherwise, only Assertion 4 is relevant. Note that a user may be permitted to view a particular entity in a CCR container if he refers to it directly, but be denied access if he refers to it indirectly. This provides a means for dealing with the aggregation problem without requiring duplicate copies of protected information: a collection of confidential aggregation-sensitive objects might be stored in a container marked secret-CCR. A user with a confidential clearance who had been given the ID of an individual object could refer to it directly, but he would be unable to view the same item via an indirect reference that identified it as a member of the secret-CCR container. Assertion 1 always requires that the user (or his role) be in the access set for the entity-operation pair specified.

- Is there anything in the system that is not (or is not part of) an entity or a user?

From the user's point of view, no. There may be structures in the implementation that the user is unaware of and that would be difficult to assign a legitimate classification to (such as internal operating system queues, perhaps). Anything the user can create, display, or modify, however, must be (or be part of) an entity or a user.

- What are the relationships among a user, an operation he invokes, and programs that the operation may invoke on his behalf? For example, what privileges do the programs inherit, and how is it determined whether a given invocation is allowed under the security policy?

A user has a clearance recorded in the system. When a user invokes an operation on an entity, his clearance and role, the appropriate device classifications, and the classification, CCR mark, and access set for that entity determine whether the operation is permitted. The user's ID or current role must be paired with the specified operation in the access set of the entity in question (Assertion 1). If the operation allows information to be viewed via a given device, then the user's clearance and the classification of the output device must equal or exceed the classification of the information (Assertion 4). Similarly, other security assertions must not be violated by the programs invoked as part of the requested operation.

- There are no integrity levels or controls defined in the model. What prevents accidental or malicious modification of sensitive data?

The reasons for omitting integrity levels have been discussed previously [16]. Modifications of clearances, classifications, and role sets are covered in the given set of assertions. Any alteration of data must presumably be accomplished by a user's invoking an operation; his authorization to invoke that operation is required by Assertion 1. In the future, specific cases may be treated in additional assertions similar to Assertion 10.

## FORMALIZING THE MMS SECURITY MODEL

To provide a firm foundation for proofs about the security properties of a system specification or implementation, a formal statement of its security model is needed. This section presents a formal model that corresponds to the informal MMS security model. It serves three purposes: it is an example of how an informal model of a system's security requirements can be made formal; being abstract, the formal model can be interpreted by others for different but related applications; and it is a basis for proofs about particular message system specifications and implementations.

The MMS security model comprises 15 definitions, a one-paragraph description of MMS operation, four assumptions about user behavior, and ten assertions that hold for the MMS. We focus on formalizing the ten assertions only, although in doing so, some notation is required to define formal entities that correspond to those discussed informally in the 15 definitions. The assertions are explicated formally in Definition 2 concerning system states and in Definitions 5 through 11 concerning the system transform. Although the correctness of the explication cannot be proven, we discuss the correspondence between the formalism and the informal model briefly following the explication.

Each MMS family member can be modeled as an automaton with inputs, an internal state, and outputs. The inputs correspond to the commands users give to the system. Because this is a *security* model, we are principally concerned with modeling the categories of inputs that affect system security. The internal state of the automaton corresponds to the information currently stored in the message system — messages, message files, classifications, access sets, and so on. Output from the automaton consists of command responses — the things that users view or obtain in response to particular requests. These may include entities, parts of entities, classification labels, and IDs. We model output as a set of distinguished entities; although output is treated as part of the internal state, it represents that part that is directly visible to users. Some commands cause a state change that affects the output set, others may cause a change of state without changing the output set, and still others (particularly commands that do not satisfy the security assertions) cause no state change at all. A history of the system is a particular sequence of inputs and states.

## Approach

We assume the existence of a set of possible users and a set of possible entities. Given these sets we define *system state* and the notion of a *secure state*. Next, we define *system* and *history* and introduce constraints on the transform that moves a system from one state to another. A system whose transform meets all these constraints is said to be *transform secure*. Finally, the notions of *secure history* and *secure system* are defined.

The structure of the formal model is intended to simplify its application to defining preconditions and postconditions for system operations. To make explicit the entities that a given operation may change, we define the concept of *potential modification* based, in part, on the work of Popek and Farber [17]. Potential modification is similar to *strong dependency*, developed by Cohen [18].

## System State

In this section we define what it is to be a system state and what it is for a system state to be secure. We assume the existence of the following disjoint sets:

**OP** is a set of operations.

**L** is a set of security levels.  $\geq$  is a partial order on  $L$  such that  $(L, \geq)$  is a lattice.

**UI** is a set of userID's.

**RL** is a set of user roles.

**US** is a set of users. For all  $u \in US$ ,  $CU(u) \in L$  is the clearance of  $u$ ,  $R(u) \subseteq RL$  is the set of authorized roles for  $u$ , and  $RO(u) \subseteq RL$  is the current role set for user  $u$ .

**RF** is a set of references. This set is partitioned into a set, **DR**, of direct references and a set, **IR**, of indirect references. Although the exact nature of these references is unimportant, we assume that the direct references can be ordered by the integers. In this model we treat each direct reference as a unary sequence consisting of a single integer, e. g.,  $\langle 17 \rangle$ . Each indirect reference is treated as a finite sequence of two or more integers, e. g.,  $\langle n_1, \dots, n_m \rangle$ , where  $\langle n_1 \rangle$  is a direct reference.

**VS** is a set of character strings. These strings serve primarily as entity values (e. g., file or message contents).

**TY** is a set of message system data types that includes **DM** for draft messages and **RM** for released messages.

**ES** is a set of entities. For all  $e \in ES$ ,  $CE(e) \in L$  is the classification of  $e$ .  $CCR(e)$  is *true* iff  $e$  is marked **CCR**, otherwise *false*.  $AS(e) \subseteq (UI \cup RL) \times OP \times N$  is a set of triples that compose the access set of  $e$ .  $(u, op, k) \in AS(e)$  iff  $u$  is a userID or user role authorized to perform operation  $op$  with a reference to  $e$  as  $op$ 's  $k$ th parameter.  $T(e) \in TY$  is the type of entity  $e$ .  $V(e) \in VS$  is the value of entity  $e$ . If  $T(e) = DM$  or  $T(e) = RM$ , then  $V(e)$  includes a releaser field  $RE(e)$ , which if nonempty contains a userID.

$ES$  contains as a subset the set of entities that are containers. For any entity  $e$  in this set,  $H(e) = \langle e_1, \dots, e_n \rangle$ , where entity  $e_i$  is the  $i$ th entity contained in  $e$ . The set  $O$  of output devices is a subset of the set of containers.<sup>†</sup> Elements  $o \in O$  serve as the domain of two further functions.  $D(o)$  is a set of ordered pairs  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , where each  $y_i$  is displayed on  $o$ . Each  $x_i$  is either a user or an entity, and the corresponding  $y_i$  is either a reference or the result of applying one of the above functions to  $x_i$ .<sup>‡</sup> We require that  $(x, V(x)) \in D(o) \rightarrow x \in H(o)$ .<sup>§</sup>  $CD(o)$  gives the maximum classification of information that may be displayed on  $o$ . This allows  $CE(o)$  to be used as the current upper limit of the classification of information to be displayed by the output device, so that users can restrict the classification of output to be lower than the maximum level permitted.

A state maps a subset of userIDs and references (intuitively, those that exist in the state in question) to elements of  $US$  and  $ES$  that represent their corresponding properties. A state also maps a subset of userIDs that "exist" into references that correspond to output devices (intuitively, these users are logged on to the specified devices). To this end we define three mappings. An *id function*,  $U$ , is a one-to-one mapping from a (possibly improper) subset of  $UI$  into  $US$ . A *reference function*,  $E$ , is a mapping from a (possibly improper) subset of  $RF$  into  $ES$  such that, for all  $n \geq 2$ ,  $E(\langle i_1, \dots, i_n \rangle) = e$  iff  $E(\langle i_1, \dots, i_{n-1} \rangle) = e^*$ , where  $e^*$  is a container such that  $e$  is the  $i_n$ th element of  $H(e^*)$ . For any reference  $r$ , if  $E(r) = e$ , we say that  $r$  is a reference to  $e$  (relative to  $E$ ). A *log-in function*,  $LO$ , is a one-to-one mapping from a (possibly improper) subset of  $UI$  into  $RF$ .<sup>\*\*</sup>

Given a reference function,  $E$ , each indirect reference of the form  $\langle n_0, \dots, n_m \rangle$  to an entity  $e_m$  corresponds to a path of entities  $\langle e_0, \dots, e_m \rangle$  such that each  $e_i \in \text{rng}(E)$ ,  $e_0$  is denoted by the direct reference  $\langle n_0 \rangle$ , and for all positive integers  $i \leq m$ ,  $e_i$  is the  $n_i$ th entity in container  $e_{i-1}$ . Such an indirect reference is said to be *based on* each entity  $e_j$  where  $0 \leq j < m$ .

**Definition 1:** A *system state*,  $s$ , is an ordered triple<sup>††</sup>  $(U, E, LO)$ , where  $U$  is an id function,  $E$  is a reference function, and  $LO$  is a log-in function such that  $\text{dom}(LO) \subseteq \text{dom}(U)$  and  $\text{rng}(LO) \subseteq \text{dom}(E \cap (RF \times O))$ . We also require that if  $o \in \text{rng}(E) \cap O$  and  $(x, y) \in D(o)$ , then  $x \in \text{rng}(E) \cup \text{rng}(U)$  to assure that only information about users and entities that "exist" in the current state can actually be displayed, and that for any reference  $r$ ,  $(x, r) \in D(o) \rightarrow E(r) = x$ . Finally, we require that  $E(LO(u_1)) = E(LO(u_2)) \rightarrow u_1 = u_2$  to prevent two users from being logged in simultaneously on the same terminal.

Given a system state  $s = (U, E, LO)$ , we abbreviate  $E(r)$  by  $r_s$ ,  $U(u)$  by  $u_s$ , and  $E(LO(u))$  by  $\hat{u}_s$ .

**Definition 2:** A state  $s$  is *secure* if  $\forall x, y \in \text{rng}(E)$ ,  $\forall o \in O \cap \text{rng}(E)$ ,  $\forall w \in \text{dom}(LO)$ , and  $\forall u \in \text{rng}(U)$ :

$$x \in H(y) \rightarrow CE(x) \leq CE(y),$$

$$x \in H(\hat{w}_s) \rightarrow CU(w_s) \geq CE(x),$$

<sup>†</sup>In implementations, some kinds of output "disappear" from the system state (e.g., information sent to a printer or a telecommunications port) while others persist (e.g., information displayed on the screen of a terminal, which a user may later refer to and modify). In the formalization, we do not distinguish between these types; both are intended to be covered by  $O$ .

<sup>‡</sup>Both the item and what is displayed must be specified so that cases in which, for example, two entities have identical values but different security levels can be distinguished.

<sup>§</sup>We extend the set theoretic notions of membership and intersection to apply to tuples in the obvious sense.

<sup>\*\*</sup>The condition that  $LO$  is a function reflects an assumption that a user cannot be on two terminals at the same time. This assumption is merely for ease of exposition.

<sup>††</sup>State is defined as a tuple, rather than as a set of functions, because two states whose elements have the same values are in fact identical, while two entities for which the defined functions return the same values may in fact be different (e.g., two copies of the same citation).

$$(x, V(x)) \in D(o) \rightarrow (x, CE(x)) \in D(o),$$

$$RO(u) \subseteq R(u), \text{ and}$$

$$CD(o) \geq CE(o).$$

### Secure System

In this section we define what a system is and what it is for a system to be secure.

**Definition 3:** A system  $\Sigma$  is a 4-tuple  $(I, S, s_0, T)$  where

$I$  is the set of well-formed system requests, where each request  $i \in I$  is of the form  $\langle op, x_1, x_2, \dots, x_n \rangle$ , where each  $x_j \in RF \cup UI \cup VS$  and  $op \in OP$ ,

$S$  is the set of possible system states;

$s_0$  designates a special state called the *initial state*, and

$T$  is the *system transform*, i.e., a function from  $UI \times I \times S$  into  $S$ .

**Definition 4:** A *history*,  $\Pi$ , of a system is a function from the set of nonnegative integers  $N$  to  $UI \times I \times S$  such that (1) the third element of  $\Pi(0)$  is  $s_0$  and (2) for all  $n \in N$ , if  $\Pi(n) = (u, i, s)$  and  $\Pi(n+1) = (u^*, i^*, s^*)$ , then  $T(u, i, s) = s^*$ .

Before defining what it means for an operation to potentially modify an entity, we notice that a reference function  $E$  and *a fortiori* a state  $S$  induce a set of functions defined on references that are counterparts to the set of functions introduced previously that are defined on entities. For example, there is a function, call it  $V_s$ , such that  $V_s(r) = V(r_s)$ . Similarly, there is a counterpart relation, call it  $H_s$ , such that  $H_s \langle r, r^1, \dots, r^n \rangle$  iff  $H(r_s) = \langle r_s^1, \dots, r_s^n \rangle$ . Each counterpart is the user-visible version of the corresponding entity function. We call these *referential counterparts* and use them to define what it means for two states to be equivalent except for a set of references. (We could have developed the entire formal model in terms of referential counterparts, but preferred the simplicity of functions to working with the relations  $H_s$  and  $LO_s$ .)

States  $s = (U, E, LO)$  and  $s^* = (U^*, E^*, LO^*)$  are *equivalent except for some set of references*  $S$  iff (1)  $U = U^*$ , (2)  $LO = LO^*$ , (3)  $dom(E) = dom(E^*)$ , (4) for any entity function  $F$  except  $V, F_s = F_{s^*}$ , and (5) for any reference  $r \in dom(E) \sim S$ ,  $V_s(r) = V_{s^*}(r)$ .

We now define potential modification as follows:

$u, i, s$  *potentially modify*  $r$  iff  $\exists s_1, s_1^*$ :  $s_1$  is equivalent to  $s$  except possibly for some set of references and  $T(u, i, s_1) = s_1^*$  and  $V(r_{s_1}) \neq V(r_{s_1^*})$ .

Call  $y$  a *contributing factor* in such a case iff  $\exists s_1$  as above and  $s_2, s_2^*$ :  $s_1$  and  $s_2$  are equivalent except for  $\{y\}$  and  $T(u, i, s_2) = s_2^*$  and  $V(r_{s_2}) \neq V(r_{s_2^*})$ .

<sup>\*</sup>This covers cases of creation (and deletion), since  $V(r_{s_1})$  will be undefined and  $V(r_{s_1^*})$  will be defined (although possibly empty).

That is,  $u, i, s$  potentially modifies  $r$  if there is some (second) state that may differ from  $s$  in the values of some entities, and  $T$  maps  $u, i$ , and this state into a third state in which  $r$ 's value differs from that which it had in the second state. The contributing factors are those entities whose values affect  $r$ 's final value.

For each referential counterpart and each function defined on users, we posit a unique operation that changes an entity or user with respect to that function. For example, an operation  $set\_AS(r, new\_access\_set)$  is the *only* operation that affects  $r$ 's access set, and it has no other user-visible effect. Further, if the transition is, e. g., from state  $s$  to state  $s^*$ ,  $AS_{s^*}(r)$  is  $new\_access\_set$  if  $new\_access\_set$  is a character string and  $V_s(new\_access\_set)$  if  $new\_access\_set$  is an entity reference. Changes to the domain of  $E$  or  $U$  (creation or deletion of entities or users) are also assumed to occur only by explicit request. The formal release operation defined later is the single exception to this assumption; it changes the *type* of  $r$  and, potentially, the releaser field of  $r$ 's value as well.

The exact nature of these operations is unimportant, since these assumptions are included solely for ease of exposition. Their purpose is not to rule out implementation commands that affect different parts of entities, but to eliminate the problem of unspecified side effects in the formal model (e. g., permission to view a message marked CCR is not permission to clear the CCR mark). Implementation commands that can alter more than a single part of a single entity correspond to a sequence of formal operations. For a given implementation, this correspondence is determined by the semantics of the implementation command language. Once this correspondence has been determined, so that the security-relevant effects of each user command are clear,  $I$  can be replaced by the set of implementation commands, with access sets also changed accordingly. Nevertheless, prudence dictates that modifications that can be made only by the *security officer* (e. g., changing a user's clearance) be restricted so that there is only a single command that performs them in any implementation.

The following constraints on the system transform lead to the definition of a *secure history* and a *secure system*. Where quantification is not explicit, universal quantification is assumed.

**Definition 5:** A transform  $T$  is *access secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*, [(op \in i \cap OP \text{ and } r_k \in i \cap RF) \rightarrow ((u, op, k) \in AS(E(r_k)) \text{ or } \exists l \in RO(u_s) \text{ and } (l, op, k) \in AS(E(r_k))))] \text{ or } s = s^*$ .

**Definition 6:** A transform  $T$  is *copy secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*$ ,  
 $x$  is potentially modified with  $y$  as a contributing factor  $\rightarrow CE(x_s) \geq CE(y_s)$ .

**Definition 7:** A transform  $T$  is *CCR secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*$ ,  
 $r \in i \cap IR$  is based on  $y$  and  $CCR(y)$ , and  $z$  is potentially modified with  $r$  as a contributing factor  $\rightarrow CU(u_s) \geq CE(y)$ .

**Definition 8:** A transform  $T$  is *translation secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*$ ,  $x \in DR$  and  $(x_s, x) \in D(\hat{u}_s)$   
 $\rightarrow \exists r \in i \cap RF, r_s = x_s$  and  $(r \text{ is based on } z \text{ and } CCR(z) \rightarrow CU(u_s) \geq CE(z))$ .

**Definition 9:** A transform  $T$  is *set secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*$ ,  
 (a)  $\exists o \in dom(E \cap (RF \times O)), CD(o_s) \neq CD(o_{s^*})$  or  $\exists x \in dom(U), CU(x_s) \neq CU(x_{s^*})$  or  $R(x_s) \neq R(x_{s^*}) \rightarrow security\_officer \in RO(u_s)$ ; and  
 (b)  $x \in dom(U)$  and  $RO(x_s) \neq RO(x_{s^*}) \rightarrow u_s = x_s$  or  $security\_officer \in RO(u_s)$ .

**Definition 10:** A transform  $T$  is *downgrade secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*$ ,  $x \in dom(E \sim (RF \times \{\hat{u}_s\}))$   
 and  $CE(x_s) > CE(x_{s^*}) \rightarrow downgrader \in RO(u_s)$ .

<sup>2</sup>For simplicity we disregard *error messages* in the formalism. In an implementation we assume that if an unauthorized operation is attempted an appropriate error message will be produced in the next state.

**Definition 11:** A transform  $T$  is *release secure* iff  $\forall u, i, s, s^*: T(u, i, s) = s^*, (T(x_s) = RM \rightarrow T(x_{s^*}) = RM$  and  $RE(x_{s^*}) = RE(x_s)$  and  $(T(x_s) \neq RM$  and  $T(x_{s^*}) = RM \rightarrow RE(x_{s^*}) = u, \exists r: r_s = x_s, i$  is the operation  $\langle \text{release}, r \rangle$ ,  $\text{releaser} \in RO(u_s)$  and  $T(x_s) = DM)$ .

**Definition 12:** A transform is *transform secure* iff it is access secure, copy secure, CCR secure, translation secure, set secure, downgrade secure, and release secure.

**Definition 13:** A history is *secure* if all its states are state secure and its transform is transform secure.

**Definition 14:** A system is *secure* if each of its histories is secure.

### Discussion

Perhaps the most basic decision we made in formalizing the MMS model concerned our general conception of a computer system, in particular the relation between a system state and a system. We considered a view where a system state consists of entities and their relations, and a system adds to this users and user operations on entities. Hence, all restrictions on user properties (in particular, the restriction that for all  $u$ ,  $RO(u) \subseteq R(u)$ ) are included in the definition of a secure system. We chose instead to view the distinction between system states and systems in terms of static as opposed to dynamic properties. Static properties are those that hold for all secure states and, hence, can be checked by examining a state in isolation; dynamic properties are those that need only hold for the relation between secure states and, hence, can be checked only by comparing two or more states. In the view we adopted, all static security properties are included in the definition of a secure state.

To a large extent the choice in conceptualizations is a matter of taste. Bell and LaPadula [4] use the latter, while Feiertag et al. [5] lean to the former. By minimizing the notion of a secure state, the former view makes the basic security theorem shorter. The deciding factor in our adopting the latter view is that it makes it impossible for a system to undergo a security-relevant change without undergoing a change in state.

Principal difficulties we encountered in formalizing the MMS security model were in representing "copy" and "view," system output, and the notion of an authorized operation. Assertion 3 (Changes to objects) in the informal model requires formal semantics to reflect the movement of information between entities, while Assertion 4 (Viewing) requires formal semantics to reflect making an entity visible to a user. Assertion 5 (Accessing CCR entities) now addresses both copying and viewing. The semantics for "copy," embodied in the definitions of "potential modification" and "contributing factor," are based on a broad interpretation of "copy." Information is considered to be copied, not only if it is directly moved from one entity to another, but also if it contributes to the potential modification of some other entity. For example, if an operation scans message file A and copies messages selected by a filter F to message file B, both A and F contribute to the potential modification of B (and are therefore subject to the constraints imposed by *copy secure* and *CCR secure*), even if both A and F are empty. The semantics for "view" are straightforward: a thing is viewed if an operation makes it a member of an output container. In light of these considerations, we have used "access" instead of "view" in Assertion 5.

In the formalization, system output is interpreted as a set of containers; other entities, parts of entities, references, and classifications that are made visible to a user are interpreted as being copied to the user's output container. We assume that in any implementation the classifications displayed appear close to the entities (or parts) they correspond to, but we have not formalized this assumption. References are explicitly included as a part of output, because the same operation applied to the same entities



can yield different results depending on how the entities are referenced. This leads to the constraint (*translation secure*) on operations that produce as output direct references that are translations of indirect ones. To enforce this constraint, the system must recognize references as a particular kind of output.

Formalizing the concept of an authorized operation is difficult because the semantics of authorized operations are unspecified. Our definition of *access secure* requires that, if an operation changes the system state (beyond producing an error message as output), then for each entity in the set of operands, the user or role, operation, and operand index must appear in the access set. Unauthorized operations must not alter the system state except to report that they are erroneous.

### Correspondence to the Informal Model

Assertions 2, 4, and 7 of the informal model, concerning classification hierarchy, viewing, and labeling, are incorporated in the formal definition of *secure state*. They correspond respectively to the first three conditions a secure state must satisfy; the last two conditions require that each user's current role set must be a subset of his authorized role set and that the current security level of each output device must be dominated by its maximum allowed level. These last two conditions are implicit in the informal model.

The remaining assertions of the informal model have been translated into constraints on the system transform. Assertion 1 (authorization) corresponds directly to *access secure*, Assertion 6 to *translation secure*, and Assertions 8 through 10 (setting, downgrading, and releasing) correspond respectively to *set secure*, *downgrade secure*, and *release secure*. *Set secure* restricts the permission to set device classifications and user role sets to security officers and restricts permission to set a user's current role to that user or a security officer. *Downgrade secure* contains an exception for  $\hat{u}$ , so that a user is not prohibited from lowering the current level of his output-device. The formal statement of *release secure* makes explicit the requirement that, once released, a message cannot have its type or releaser field altered.

Assertions 3 and 5 correspond to *copy secure* and *CCR secure*. The definition of *copy secure* actually covers parts of both Assertion 3 and Assertion 4, because output devices are treated as containers. So if entity  $x$  receives information from an object  $y$ ,  $CE(x) \geq CE(y)$  (Assertion 3); and if an output container  $o$  receives information from entity  $x$ ,  $CE(o) \geq CE(x)$  (Assertion 4). *CCR secure* corresponds to Assertion 5, under the interpretation that having access to an entity is significant only if that entity is a contributing factor in the potential modification of another entity.

### Storage Channels

Because we have defined *potential modification* and *contributing factor* in terms of changes only to the *value* of an entity, the constraints imposed by *copy secure* and *CCR secure* do not apply to changes made to other functions defined on entities (classification, CCR mark, access set, and type) or those defined on users (clearance, role set, and current role). Thus, information could be transferred from a higher level to a lower one through these functions. However, changes to user clearances and role sets are controlled by *set secure*; changes to classifications are controlled by *downgrade secure*; changes to entity type are generally restricted severely by the semantics of the message system; and changing a CCR mark provides only a single bit of information at a time. Changes to the access set for an entity could provide a higher rate of information transfer, and if this were a serious concern, the definitions in question could be modified to include changes of this kind. We chose not to include these because our principal concern is with changes to the values of entities, and the added notational complexity would only cloud the presentation. We have left for others the problem of dealing with classifications that are themselves classified, such as highly sensitive compartment names.

### A Basic Security Theorem for the Formal MMS Security Model

In formalizations where a secure system is a collection of secure states, some feel that a Basic Security Theorem is needed to show the restrictions on system transforms that insure that a system starting in a secure state will not reach a state that is not secure [4]. Such theorems are of little practical significance, since their proofs do not depend on the particular definition of security provided by the model [19]. Further, in our approach such a theorem is not pressing, since the concept of a secure system is defined largely in terms of a secure transform. Nevertheless, we do appeal to the notion of a secure state, and some readers may feel that some form of basic security theorem is needed. Those readers should find it trivial to prove the following analog of the Basic Security Theorem for our definition of a secure state.

**Theorem:** Every state of a system  $\Sigma$  is secure if  $s_0$  is secure and  $T$  meets the following conditions for all  $u, i, s, s^*: T(u, i, s) = s^*$  and for all  $x, y \in RF, w \in US$ :

1.  $x_i \notin H(y_i)$  and  $x_{i^*} \in H(y_{i^*}) \rightarrow CE(x_{i^*}) \leq CE(y_{i^*})$ .
2.  $x_i \in H(y_i)$  and  $CE(x_{i^*}) \not\leq CE(y_{i^*}) \rightarrow x_{i^*} \notin H(y_{i^*})$ .
3.  $x_i \notin H(\hat{w}_i)$  and  $x_{i^*} \in H(\hat{w}_{i^*}) \rightarrow CU(w_{i^*}) \geq CE(x_{i^*})$ .
4.  $x_i \in H(\hat{w}_i)$  and  $CU(w_{i^*}) \not\geq CE(x_{i^*}) \rightarrow x_{i^*} \notin H(\hat{w}_{i^*})$ .
5.  $(x_i, V(x_i)) \notin \hat{w}_i$  and  $(x_{i^*}, V(x_{i^*})) \in \hat{w}_{i^*} \rightarrow (x_{i^*}, CE(x_{i^*})) \in \hat{w}_{i^*}$ .
6.  $(x_i, V(x_i)) \in \hat{w}_i$  and  $(x_{i^*}, CE(x_{i^*})) \notin \hat{w}_{i^*} \rightarrow (x_{i^*}, V(x_{i^*})) \notin \hat{w}_{i^*}$ .
7.  $R(w_i) \not\subseteq R(w_{i^*})$  or  $RO(w_i) \not\subseteq RO(w_{i^*}) \rightarrow RO(w_{i^*}) \subseteq R(w_{i^*})$ .
8.  $CE(\hat{w}_i) \not\subseteq CE(\hat{w}_{i^*})$  or  $CD(\hat{w}_i) \not\subseteq CD(\hat{w}_{i^*}) \rightarrow CD(\hat{w}_{i^*}) \supseteq CE(\hat{w}_{i^*})$ .

Together, conditions 1 through 8 are necessary and sufficient conditions for every state of a system to be secure in any system that does not contain states that are unreachable from  $s_0$ .

### OTHER ELEMENTS OF OUR APPROACH

At the Naval Research Laboratory we are producing, in addition to the security model, a requirements document, a software design, and two full-scale prototypes of future military message systems (MMSs). One of the two prototypes will be designed for a military environment where little message handling is required (e.g., a submarine) and thus will perform few operations. The other prototype will perform a larger set of operations. The intent of the two prototypes is to demonstrate the viability of our approach to building secure systems. In addition, the prototypes can serve as the basis for production-quality systems. The security model, based on the message-system application, was described in previous sections. Other elements of our approach are described here.

#### Family Methodology

Review of the requirements of future military message systems suggests that one system will not suffice for all environments [8]. Several similar systems will be needed that share certain functions and enforce military security. Because they will operate in different environments, these systems will have important differences, e.g., in their user command languages, in the operating systems that they use, and in the hardware on which they are implemented.

The design problem is to exploit the similarities of these systems without unduly constraining the individual variations that make each message system suitable for its particular environment. A message system for intelligence analysts, for example, shares certain features with a message system for command-center personnel, but it may be unreasonable to insist that intelligence analysts and command-center personnel use exactly the same system.

To deal with this problem, we have adopted the family methodology [20,21]. This methodology requires developers to consider the entire family before building a single member. We are applying the methodology to two major products: a requirements document that applies to all members and a design document that defines a modular structure suitable for all members. In the modular structure, the different features of family members are assigned to separate modules, so that different family members can be produced by replacing modules without changing the overall program structure [22]. For example, two family members that are identical except for their user command languages will differ only in their implementations of the user command language module.

### **Requirements Document**

A major product of the project is a requirements document. Because its goal is to describe the shared features of family members, the document excludes decisions about features that differentiate members, e.g., decisions about user command languages, computer hardware, and operating systems. Included in the document are a set of operations (i.e., user services), where each family member is associated with some subset, and the security model described previously, which applies to all members. Our requirements document adheres to the guidelines given by Heninger [23]: it describes what is required without making design decisions; and it attempts to provide a precise, consistent, and complete description of the requirements.

A central feature of the document is its use of an intermediate command language (ICL) [24], the union of the sets of operations required by family members. The purpose of the ICL is to express the user-visible behavior of family members in an abstract way. The ICL is partitioned into several groups so that all ICL commands within a single group are associated with the same data type. In the terms of the MMS security model, each ICL command that can affect security corresponds approximately to one or more operations, and the set of security-relevant ICL commands for a family member corresponds to the transform for that system.

Each family member is associated with a subset of the ICL. Because differences in user command languages do not affect the ICL, two members that perform the same operations but have different user command languages will be associated with the same ICL subset. Given two systems in which one system performs only a subset of the operations performed by the second, the ICL subset for the smaller system will be contained in the ICL subset for the larger system.

### **Rapid Prototyping**

To evaluate our requirements document, we are building rapid prototypes [25]. These allow us to test the document for consistency, completeness, and correctness and to assess the suitability of the chosen ICL subsets for implementation as full-scale prototypes. They also allow us to see the effects the security model has on the user interface. More specifically, the rapid prototypes should help to answer the following questions:

- How can the functional specifications be improved?
- How should the security model be changed?
- How can we more closely integrate the security model and the functional specifications?

## CONCLUSIONS

We favor an approach to building secure systems that includes an application-based security model. An instance of such a model and its formalization have been presented. They are intended as examples for others who wish to use this approach. Important aspects of the model are summarized as follows:

- Because it is framed in terms of operations and data objects that the user sees, the model captures the system's security requirements in a way that is understandable to users.
- The model defines a hierarchy of entities and references; access to an entity can be controlled based on the path used to refer to it.
- Because the model avoids specifying implementation strategies, software developers are free to choose the most effective implementation.
- The model and its formalization provide a basis for certifiers to assess the security of the system as a whole.

Simplicity and clarity in the model's statement have been primary goals. The model's statement does not, however, disguise the complexity that is inherent in the application. In this respect, we have striven for a model that is as simple as possible but stops short of distorting the user's view of the system.

The work reported here demonstrates the feasibility of defining an application-based security model informally and subsequently formalizing it. The security model described has been used almost without change by another message system project [26], and it has been adapted for use in document preparation and bibliographic systems [27].

Judgments about the viability of our approach as a whole must await its application in building full-scale systems. This we are pursuing in the development of message-system prototypes.

## ACKNOWLEDGMENTS

Many individuals contributed to the work reported here. Discussions with D. Parnas led to an initial version of the security model. Later revisions of the model were based on reviews by S. Wilson, M. Cornwell, R. Jacob, J. Miller, M. Schaefer, J. Millen, and others too numerous to mention. Participants in the 1982 Air Force Summer Study on Multi-Level Data Management Security also provided many helpful comments. For providing the funding that allows us to continue our work, we are grateful to H. O. Lubbes of the Naval Electronic Systems Command and to the Office of Naval Research.

## REFERENCES

1. C. E. Landwehr, "Formal Models for Computer Security," *ACM Computing Surveys* 13(3), 247-278 (Sept. 1981).
2. C. E. Landwehr and C. L. Heitmeyer, "Military Message Systems: Requirements and Security Model," NRL Memorandum Report 4925, Sept. 1982.
3. C. E. Landwehr, "Assertions for Verification of Multilevel Secure Military Message Systems," *ACM SIGSOFT Software Eng. Notes* 5(3), 46-47 (July 1980).

4. D. E. Bell and L. J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation," M74-244, MITRE Corp., Bedford, Massachusetts, July, 1975.
5. R. J. Feiertag, K. N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," in *Proceedings of the 6th ACM Symposium on Operating Systems Principles, ACM SIGOPS Operat. Syst. Rev.* 11(5), 57-65 (Nov. 1977).
6. E. J. McCauley and P. J. Drongowski, "KSOS — The Design of a Secure Operating System," in *Proceedings of the AFIPS National Computer Conference*, Vol. 48, June 1979, pp. 345-353.
7. S. R. Ames, Jr., and D. R. Oestreicher, "Design of a Message Processing System for a Multilevel Secure Environment," in *Proceedings of the AFIPS National Computer Conference*, June 1978, pp. 765-771.
8. C. L. Heitmeyer and S. H. Wilson, "Military Message Systems: Current Status and Future Directions," *IEEE Trans. Commun.* COM-28, 1645-1654 (Sept. 1980).
9. T. H. Myer and C. D. Mooers, "HERMES User's Guide," Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, Feb. 1977.
10. D.E. Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," MTR-2547, Vol. III, MITRE Corp., Bedford, Massachusetts, Apr. 1974, pp.30,31. (AD-780-528)
11. S. H. Wilson, N. C. Goodwin, E. H. Bersoff, and N. M. Thomas III, "Military Message Experiment — Vol. I. Executive Summary," NRL Memorandum Report 4454, Mar. 1982.
12. J. Rothenberg, "SIGMA Message Service: Reference Manual, Version 2.3," Report ISI/TM-78-11.2, University of Southern California Information Sciences Institute, Marina del Rey, California, June 1979. (AD-A072-840)
13. R. Stotz, R. Tugender, and D. Wilczynski, "SIGMA — An Interactive Message Service for the Military Message Experiment," in *Proceedings of the AFIPS National Computer Conference*, Vol. 48, June 1979, pp. 839-853.
14. J.P.L. Woodward, "Applications for Multilevel Secure Operating Systems," in *Proceedings of the AFIPS National Computer Conference*, Vol. 48, June 1979, pp.319-328.
15. D. E. Denning, "A Lattice Model of Secure Information Flow," *Commun. ACM* 19(5), 236-243 (May 1976).
16. C. E. Landwehr, "What Security Levels Are for and Why Integrity Levels Are Unnecessary," NRL Technical Memo 7590-308:CL:uni, Feb. 1982.
17. G. J. Popek and D. A. Farber, "A Model for Verification of Data Security in Operating Systems," *Commun. ACM* 21, 737-749 (Sept. 1978).
18. E. Cohen, "Information Transmission in Computational Systems," in *Proceedings of the 6th ACM Symposium on Operating Systems Principles, ACM SIGOPS Oper. Syst. Rev.* 11(5), 133-139 (Nov. 1977).
19. J. McLean, "A Comment on the Basic Security Theorem of Bell and LaPadula," submitted for publication, 1983.

20. D. L. Parnas, "On the Design and Development of Program Families," *IEEE Trans. Software Eng.* SE-2, 1-9 (Mar. 1976).
21. D. L. Parnas, "Designing Software for Ease of Extension and Contraction," *IEEE Trans. Software Eng.* SE-5, 128-138 (Mar. 1979).
22. D. L. Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules," *Commun. ACM* 15, 1053-1058 (Dec. 1972).
23. K. L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Application," *IEEE Trans. Software Eng.* SE-6, 2-13 (Jan. 1980).
24. C. L. Heitmeyer, "An Intermediate Command Language for Military Message Systems," NRL Technical Memo 7590-450:CH:ch, Nov. 1981.
25. C. Heitmeyer, C. Landwehr, and M. Cornwell, "The Use of Quick Prototypes in the Secure Military Message Systems Project," *ACM SIGSOFT Software Eng. Notes* 7(5), 85-87 (Dec. 1982).
26. Harry C. Forsdick and Robert H. Thomas, "The Design of Diamond—A Distributed Multimedia Document System," BBN Report 5204, Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, Oct. 1982.
27. Air Force Studies Board, Commission on Engineering and Technical Systems, National Research Council, "Multilevel Data Management Security" (National Academy Press, Washington, D.C., 1983).